Reinforcement Learning: An Introduction

Summary Notes

Mrinank Sharma

April 4, 2020

1 Introduction

Elements of Reinforcement Learning

Other than the **agent** and the **environment**, there are four main subelements of an RL system:

1. The *policy* maps perceived states of the environment to actions. This could be a lookup table or a search process.
2. The *reward signal* defines the goal of a reinforement learning problem. The environment sends rewards to an agent that has the sole objective of maximising this reward.
3. The *value function* specifies what is good in the long run. The *value* of a state is the total amount of reward an agent can expect to accumulate over teh future starting from that state.
4. Some RL systems have *models* of the environment that can mimic the behaviour of the environment. Models are used for planning.

Rewards and Value Functions

Whilst rewards determine the immediate, intrisic desirability of environmental states, values indicate the long-term desirability of states after taking into account which states are likely to follow. We choose actions based on values, even though values are derived from rewards.

Evolutionary Methods

It is possible to learn policies without estimating value functions by using evolutionary methods. However, they tend to ignore much of the useful structure and are not well suited for RL tasks as they ignore relevant information, such as the states which are passed and the actions which are selected.

2 Multi-armed Bandits

> *The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that* ***evaluates*** *the actions taken rather than* ***instructs*** *by giving correct actions.*

This form of feedback creates the need for active exploration.

k-armed bandits

Definition 2.1 (k-armed Bandit) *In the k-armed bandit problem, you are repeatedly faced with a choice among k different actions. After each choice, a numerical reward is received from a* ***stationary*** *probability distribution associated with the selected action. The objective is to maximise the expected total reward over some time period.*

Value

The value of action a is the expected reward given a was chosen:

$$q_*(a) = \mathbb{E}[R_t \,|\, A_t = a]. \tag{1}$$

If we knew the values, problem solved - just pick the action with the maximum value. However, we do not know these. Denote the *estimated* value as $Q_t(a)$. A simple way to estimate the action-value is a *sample average*, which can be calculated incrementally.

Reward Variance

Note that purely greedy methods are best if there is no noise in the reward signal. With ϵ-greedy algorithms, the best value of ϵ will depend on the amount of noise, which determines the number of samples for the sample average to converge. However, if the problem is *non-stationary*, we still need to explore.

If the bandit is non-stationary, we might use the following rule for value estimates: *Non-stationary Problems*

$$Q_{n+1}(a) = Q_n(a) + \alpha_n(a)\left[R_n - Q_n(a)\right]. \tag{2}$$

It is known that convergence with probability 1 happens if the following conditions hold: *Convergence Guarantees*

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty, \text{ and } \sum_{n=1}^{\infty} \alpha_n(a)^2 < \infty. \tag{3}$$

The first condition ensures the steps are sufficiently large to overcome an initial condition and the second condition ensures that the steps become small enough to give convergence. However, step sizes which meet this conditions are rarely used in applications because they seem to converge slowly or require considerable tuning.

With this techniques, the initialisation effectively becomes an additional parameter which needs to be chosen. They can be used to provide prior knowledge and can encourage exploration, for example, by being optimistic. However, this sort of exploration is not useful for non-stationary problems (the task changes creates a renewed need for exploration).

The intuition behind UCB is that its better, when exploring, to select among the non-greedy actions which have the *potential* for being optimal. To do this, we maximise an estimate of the arm's value summed with some sort of uncertainty estimate. UCB often performs well but can be difficult to extent to RL from bandits. *UCB*

We don't have to use value based methods. Instead, we could directly learn a preference over the actions and modify our preferences to give us higher rewards e.g., increasing our preference for an action if selecting it lead to a better outcome than expected.

Definition 2.2 (Contextual Bandits) *In a contextual bandit, the agent is faced with a series of multi-armed bandit problems, each associated with some information i.e., the agent has information about which bandit problem is being faced at any timestep.* *Contextual Bandits*

Contextual bandits are intermediate between multi-armed bandits and the full RL problem; they involve learning a policy (i.e., a mapping from state information to actions) but similar to k-armed bandits, **the action selected influences only the immediate reward and not the next situation**. If we drop this constraint, we have the full RL problem.

3 Finite Markov Decision Processes

Definition 3.1 (Finite Markov Decision Process (MDP)) *A finite Markov Decision Process (MDP) is made up of:* *MDP*

- *A sequence of discrete time steps,* $t = 0, 1, 2, \ldots.$
- *A discrete set of states,* $S_t \in \mathcal{S}$.
- *A set of actions which the agent can take,* $A_t \in \mathcal{A}(S_t)$.
- *A finite set of numerical rewards,* $R_t \in \mathcal{R} \subset \mathbb{R}$. *In this case, there is a well defined discrete transition distribution:*

$$p(s', r|s, a) \triangleq Pr[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a], \tag{4}$$

which defines the ***dynamics*** *of the MDP.*

Note the Markov assumption; probabilities over the next state and reward depend only on the current state and action. This can be interpreted as a restriction on the state - the state must include information about all aspects of the past agent-environment interaction which will affect the future. *Interpreting the Markov Assumption*

Given p, we can compute the following functions of interest:

$$p(s'|s, a) \triangleq \Pr[S_t = s'|S_{t-1} = s, A_{t-1} = a], \tag{5}$$

$$r(s, a) \triangleq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a], \tag{6}$$

and

$$r(s,a,s') \triangleq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']. \tag{7}$$

Agent vs Environment

A general rule of thumb is that anything that cannot be changed *arbitarily* by the agent is considered to be outside of it and thus part of the environment. The agent-environment boundary represents the limit of the agent's absolute control, not of its knowledge.

The goal of the agent is formalised in terms of the reward signal which passes from the environment to the agent. This is one of the most distinctive features of RL.

Reward Hypothesis

> *All of what we mean by goals and purposes can be well though of as the maximisation of the expected value of the cumulative sum of a received scalar signal.*

The reward signal is not the place to impart to the agent prior knowlege about *how* to achieve a task; if we do this, the agent may only achieve subgoals without actually acheiving the real goal. A better place to impart this knowledge is the initial policy or value function. **How does this link to reward shaping?**

Discount Rates

Episodic tasks terminate at some timestep, T, whilst continuing tasks do not. In the first case, we simply use the (expected) sum of rewards whilst we introduce discounting in the infinite horizon case i.e.,

$$G_t \triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{8}$$

G_t is the *discounted return*. The closer the value of γ is to 1, the more farsighted the agent is.

Unifying Notation

We convert episodic tasks to continuing tasks by introducting a special *absorbing state* which transitions only to itself and returning zero reward. We can write:

$$G_t \triangleq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{9}$$

allowing for $T = \infty$ and $\gamma = 1$. **The book says not both - but I think that should be allowed for an episodic task with absorbing state.**

Policy

Definition 3.2 (Policy) *A policy is a mapping from states to probabilities of selecting each possible action. $\pi(a|s)$ is the distribution over actions, A_t, given the current state, S_t.*

Value Function

Definition 3.3 (Value Function) *The value function of state s under policy π is the expected return when starting in s and following π.*

$$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t | S_t = s] \tag{10}$$

Action-Value Function

Definition 3.4 (Action-Value Function) *The action-value function of action a, state s under policy π is the expected return when starting in s and following π after first performing a.*

$$q_\pi(s,a) \triangleq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{11}$$

Value functions satisfy recursive relationships.

Optimal Policies

A policy, π, is better than policy π' iff $v_\pi(s) \geq v_{\pi'}(s) \forall s \in \mathcal{S}$ with a strict inequality for at least one state. Optimal policies are better than or equal to all other policies. There may be multiple optimal policies, but they all share the same state-value function,

$$v_*(s) = \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S}. \tag{12}$$

The optimal action-value function is define analogously.

Bellman Optimality Equality

The Bellman optimality equation for the value function is:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a)[r + \gamma v_*(s')], \tag{13}$$

which expresses the intuition that the optimal value of a state is equal to the expected return of the best action from that state. Similarly, for the action-value function,

$$q_*(s,a) = \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a' \in \mathcal{A}(s')} q_*(s', a')]. \tag{14}$$

For a finite MDP, the Bellman optimality equations give a system of equations which can be solved for $v_*(s)$. Once we have the optimal value functions, it is easy to determine an optimal policy by acting greedily (or performing a one-step search). **If you use v_* to select actions in the short-term, the greedy policy is optimal in the long term since this value function takes into account reward consequences.** The optimal expected long-term return is locally and immediately available for each state so a one-step lookahead search yields optimal actions.

Note that the action-value function effectively caches the results of all of the one-step lookahead searches. This function can be maximised (with respect to the action) to find the optimal policy.

Caveats

This all sounds great. However, we run into the following problems:

- We need to accurately know the dynamics of the environment.
- We need to have enough computational resources.
- The Markov property needs to hold.

4 Dynamic Programming

Limited Utility

Dynamic Programming (DP) algorithms can be used to compute optimal policies given perfect models of the environment. However, they are important only theoretically because they are limited by imperfect models and finite compute.

Policy Evaluation

Policy evaluation computes the state-value function for policy π. We turn the Bellman equation into an update rule:

$$v_{k+1}(s) \triangleq \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_k(s)], \quad \forall s \in \mathcal{S}. \tag{15}$$

v_π is a fixed point for this update rule since it must satisfy the Bellman equation. Note that the estimate in general converges as $k \to \infty$ provided $\gamma < 1$ or eventual termination from all states under π (the same conditions which give a unique value function).

This is an *expected update*; the old value is replaced with a new value computed using previous values and expected immediate reward. In fact, all DP updates are called *expected updates* since they take the expectation over successor states rather than sampling.

Algorithm 1 Iterative Policy Evaluation

```
Input: $\pi$, the policy to be evaluated
Parameters: $\theta > 0$, a threshold determining accuracy.
Initialisation: initialise $V(s)$ arbitarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$

loop:
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do:        ▷ Sweep over states, using the most recent values of $V(s)$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$
```

If we know the dynamics, knowing the value of the policy helps us improve the policy. We can compute the value of selecting action a in state s and thereafter following π. If it is better to select a once in s and thereafter follow π, it is better to always select a in s and the new policy is an improvement.

Policy Improvement Theorem

Theorem 4.1 (Policy Improvement Theorem) *Let π and π' be any pair of determinstic policies such that, for all $s \in \mathcal{S}$,*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{16}$$

Then π' must be as good as, or better than, π i.e., it must obtain greater or equal expected return from all states $s \in \mathcal{S}$:

$$v'_\pi(s) \geq v_\pi(s). \tag{17}$$

Note that if there is any strict inequality in Eq. (16)*, there must be a strict inequality in Eq.* (17) *in* at least *one state.*

Policy Improvement

The greedy policy i.e., $\pi'(s) = \arg\max_a q_\pi(s,a)$, meets the conditions of the policy improvement theorem. The process of making a new policy which improves on an original policy by making it greedy with respect its value function is known as *policy improvement*. If policy improvement gives us a policy with the same perform as the old policy, this policy must necessarily be optimal as only the optimal policy satifies the Bellman optimality equation.

Note that similar analysis can be performed for stochastic policies.

Policy Iteration

We can repeatedly evaluate policy π to calculate v_π and then improve π to form π'. This process is guaranteed to find the optimal policy.

Algorithm 2 Policy Iteration

```
    Initialisation: initialise $V(s)$ arbitarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$.
    Initalise $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$.
    Parameters: $\theta > 0$, a threshold determining accuracy.

 1: loop:                                                        ▷ Policy Evaluation
 2:     $\Delta \leftarrow 0$
 3:     for each $s \in \mathcal{S}$ do:
 4:         $v \leftarrow V(s)$
 5:         $V(s) \leftarrow \sum_{a\in\mathcal{A}(s)} \pi(a|s) \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)[r+\gamma V(s')]$
 6:         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
 7: until $\Delta < \theta$

 8: policy-stable ← True                                         ▷ Policy Improvement
 9: for each $s \in \mathcal{S}$ do:
10:     old-action ← $\pi(s)$
11:     $\pi(s) \leftarrow \arg\max_{a\in\mathcal{A}(s)} \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)[r+\gamma V(s')]$
12:     if old-action $\neq \pi(s)$ then:
13:         policy-stable ← True
14: if policy-stable then:
15:     return $V \simeq v_*, \pi \simeq \pi*$
16: else:
17:     go to 1
```

Value Iteration

It turns out that we don't even need to perform policy evaluation until convergence. We form the *value iteration* algorithm by performing one step of policy evaluation followed by policy improvement.

Algorithm 3 Value Iteration

```
    Initialisation: initialise $V(s)$ arbitarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$.
    Initalise $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$.
    Parameters: $\theta > 0$, a threshold determining accuracy.

 1: loop:
 2:     $\Delta \leftarrow 0$
 3:     for each $s \in \mathcal{S}$ do:
 4:         $v \leftarrow V(s)$
 5:         $V(s) \leftarrow \max_{a\in\mathcal{A}(s)} \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)[r+\gamma V(s')]$
 6:         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
 7: until $\Delta < \theta$
 8: return $\pi(s) = \arg\max_a p(s',r|s,a)[r+\gamma V(s')] \simeq \pi^*(s), V(s) \simeq v_*(s)$
```

Alternative Interpretation

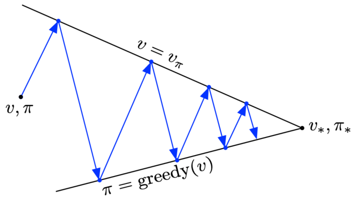
The algorithm can also be interpreted as the conversation of the Bellman optimality equality to an update rule. Above, V will converge to v_*.

Asynchronous DP

Asynchrononous DP algorithms are in-place iterative DP algorithms which are not organised in terms of sweeping the entire state space (a drawback of normal DP). The algorithm *must* continue to update all states to ensure correct convergence.

Generalised Policy Iteration

Generalied policy iteration (GPI)[1] refers to the general idea of letting policy-evaluation and policy-improvement processes interact, independent of the granularity. If **both the evaluation and improvement stabilise, the value function and policy must be optimal**. The value function stabilises only when it is consisent with the current policy and policy stabilises only when it is greedy with respect to the current value function. Both stabilising implies that a policy has been found which is greedy with respect to its own evaluation function, implying that the Bellman optimality equation holds.



Generalised Policy Iteration (GPI)

Whilst not practical for large problems, DP methods are actually quite efficient with (worst-case) complexity $\Omega(\texttt{poly}(n,k))$ where n, k are the number of states and actions respectively. They also seem to typically converge must faster than the worst case.

Bootstrapping

DP methods update estimates of the values of states based on estimates of the values of successor states; **bootstrapping**.

5 Monte Carlo Methods

Compared to DP

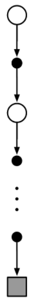
We now **relax the assumption that we have complete knowledge of the environment**, transforming the task into *learning* rather than *planning*. Whilst in DP, value functions were computed, here they will be learnt. Additionally, note that surprisingly often, even when we have perfect knowledge of the environment, it is hard to apply DP since the dynamics function, p, is difficult to obtain (though it can be easy to sample from).

Monte Carlo Methods

Monte Carlo (MC) methods solve RL problems by averaging sample returns. Here, they are defined only for episodic tasks to ensure the returns are well defined. They are incremental on an episode-by-episode sense but not in a step-by-step sense.

Monte Carlo Prediction

Monte Carlo Prediction[1] attempts to learn the state-value function for a policy. An obvious way to estimate it is to simply average the returns observed after visiting that state.



Monte Carlo Backup Diagram

Algorithm 4 First-visit MC Prediction

```
Input: policy $\pi$ to be evaluated.
Initialisation: initialise $V(s)$ arbitarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$.
Initalise returns[s] $\leftarrow$ [] for all $s \in \mathcal{S}$

loop:
    Generate an episode under $\pi$: $\{S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T\}$
    $G \leftarrow 0$
    for $t \in \{T-1, T-2, \ldots, 0\}$ do:
        $G \leftarrow R_{t+1} + \gamma G$                          ▷ Recusively compute $G$
        if $S_t \notin \{S_0, S_1, \ldots S_{t-1}\}$ then:              ▷ If first visit
            append $G$ to returns[$S_t$]
            V($S_{t+1}$) $\leftarrow$ average(returns[$S_t$])
```

The *every-visit* MC algorithm instead averages the returns following all visits to each state. Both every-visit and first-visit converge to v_π quadratically as the number of visits to each state goes to infinity.

MC methods do not bootstrap; the estimates for each state are independent. Note that the computational expense of estimating the state value is independent of the total number of states and we can generate many episodes starting from states of interest if we desire.

Exploring Starts

Note that since we no longer have a model, knowing v_π is not sufficient for policy improvement. We instead need to calculate q_π. This can be done in a manner similar to Algorithm 4 except that now each 'visit' is a tuple (s, a). However, there is no guarantee that each state-action pair is visited i.e., we have an issue of *maintaining exploration*. One way to resolve this is *exploring starts* where it is specified that the episode begins in a specific state-action pair and that every pair has a non-zero probability of selection. Alternatively, we could consider policies which always assign strictly positive probabilities to all actions.

Monte Carlo Control

We can naturally apply GPI, using MC methods to calculate q_π.

[1] Margin figure reproduced from Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

Algorithm 5 MCES: Monte Carlo Exploring Starts

```
**Initialisation:** initialise $\pi(s) \in \mathcal{A}(s)$ arbitarily, for all $s \in \mathcal{S}$. Initialise $Q(s,a)$ arbitarily
for all $s \in \mathcal{S}, a \in \mathcal{A}$. Initalise returns$[s,a]$ $\leftarrow$ [] for all $s \in \mathcal{S}$

loop:
    Choose $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability $> 0$
    Generate an episode from $S_0, A_0$ under $\pi$: $\{S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T\}$
    $G \leftarrow 0$
    for $t \in \{T-1, T-2, ..., 0\}$ do:
        $G \leftarrow R_{t+1} + \gamma G$                                          ▷ Recusively compute $G$
        if $(S_t, A_t) \notin \{(S_0, A_0), (S_1, A_0), \dots (S_{t-1}, A_{t-1})\}$ then:         ▷ If first visit
            append $G$ to returns$[S_t, A_t]$
            Q$(S_{t+1}, A_t)$ $\leftarrow$ average(returns$[S_t]$)
            $\pi(S_t) \leftarrow \arg\max_{a \in \mathcal{A}(S_t)} Q(S_t, a)$
```

Note that all returns for each state-action pair are averaged regardless of the policy used. MCES cannot converge to any suboptimal optimal; if it did, the value function would eventually converge to the correct value function, which would cause the policy to change if it is suboptimal. Additionally, it seems that updates to Q ought to decrease over time. However, formal correct convergence has not been proved for this algorithm.

Relaxing Exploring Starts

Exploring starts cannot be relied upon in general, especially when learning directly from actual interaction with an environment. We need to relax this, which we can do by, in general, allowing the agent to continue selecting them.

Algorithm 6 On-policy first-visit MC Control

```
**Parameter:**
    small $\epsilon > 0$

**Initialisation:**
    $\pi$ as an arbitarily $\epsilon$-soft policy.
    $Q(s,a)$ arbitarily for all $s \in \mathcal{S}, a \in \mathcal{A}$.
    returns$[s,a]$ $\leftarrow$ [] for all $s \in \mathcal{S}$

loop:
    Generate an episode from under $\pi$: $\{S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T\}$
    $G \leftarrow 0$
    for $t \in \{T-1, T-2, ..., 0\}$ do:
        $G \leftarrow R_{t+1} + \gamma G$                                          ▷ Recusively compute $G$
        if $(S_t, A_t) \notin \{(S_0, A_0), (S_1, A_0), \dots (S_{t-1}, A_{t-1})\}$ then:         ▷ If first visit
            append $G$ to returns$[S_t, A_t]$
            Q$(S_{t+1}, A_t)$ $\leftarrow$ average(returns$[S_t]$)
            $A^* \leftarrow \arg\max_{a \in \mathcal{A}(S_t)} Q(S_t, a)$                       ▷ Break ties arbitarily
            $\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon / \mathcal{A}(S_t), & a = A^* \\ \epsilon / \mathcal{A}(S_t), & \text{otherwise} \end{cases}, \quad \forall a \in \mathcal{A}(S_t)$
```

The policy improvement theorem ensures that any ϵ-greedy policy is an improvement over any ϵ-soft policy. Policy iteration works for ϵ-soft policies, and the above algorithm will find the best possible soft policy.

Off-Policy Learning

Learning control methods aim to learn action values conditional on subsequent optimal behaviour but they need to behave non-optimally to explore all actions. Algorithm 6 is a compromise, learning action-values for a 'near'-optimal policy. Alternatively, we could use two policies; one to generate behaviour (the *behavioural policy*) and one to approach optimal behaviour (the *target policy*). This approach is known as *off-policy learning*.

The *prediction* policy seeks to estimate q_π or v_π but from behaviour generated by policy b. We need to make the assumption of **coverage**, $\pi(a|s) > 0 \Rightarrow b(a|s) > 0$. We do off-policy learning most frequently using *importance sampling* (IS). Sparing the derivation, the *ordinary importance sampling* estimator is:

Ordinary Importance Sampling

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\
&\simeq \sum_{t\in\mathcal{T}(s)} \frac{\rho_{t:T(t)-1}}{|\mathcal{T}(s)|} G_t
\end{aligned}
\tag{18}
$$

where, for notational convenience, the episodes are appended to one another so that a timestep uniquely indentifies a timestep within an episode, $\mathcal{T}(s)$ is the set of times at which s was visited, $T(t)$ is the termination time of the episode which t belong to. $\rho_{t:T-1}$ are the importance weights:

$$
\begin{aligned}
\rho_{t:T-1} &= \prod_{k=t}^{T-1} \frac{p(R_{k+1},S_{k+1}|S_k,A_k)\pi(A_k|S_k)}{p(R_{k+1},S_{k+1}|S_k,A_k)b(A_k|S_k)} \\
&= \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.
\end{aligned}
\tag{19}
$$

The above formulae are straightforward to derive following the standard importance sampling derivation. Note that the dynamics of the unknown MDP cancel out.

Weighted Important Sampling

An alternative to Eq. (20) is *weighted importance sampling*:

$$
v_\pi(s) \simeq \sum_{t\in\mathcal{T}(s)} \frac{\rho_{t:T(t)-1}}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}} G_t. \tag{20}
$$

The first-visit version of the ordinary IS estimator is unbiased with high variance (since the variance of the importance weights can be unbounded) whilst the first-visit version of the weighted IS estimator is biased but with much lower variance: the maximum weight on any of the returns is 1. In fact, assuming bounded returns, the variance of the weighted IS estimator converges to zero even if the importance weights have infinite variance. **In practice, the weighted estimator has much lower variance and is prefered**. Additionally, note that the weighted importance sampling estimator produces a weighted average of only the returns consisent with the target policy whilst the normal importance sampling estimate does not.

First vs. Every Visit

The every-visit methods introduce bias, but the bias asymtoticallty falls to zero as the number of samples increases. These methods are prefered as they remove the need to track which states have been visited and they are easier to extend.

Algorithm 7 Off-policy Every-Visit MC Prediction

```
Input:
    arbitary target policy $\pi$

Initialisation:
    $Q(s,a)$ arbitarily for all $s \in \mathcal{S}, a \in \mathcal{A}$.
    $C(s,a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$

loop:
    $b \leftarrow$ any policy which coverage of $\pi$.
    Generate an episode from under $b$: $\{S_0,A_0,R_1,S_1,A_1,R_2,...,S_{T-1},A_{T-1},R_T\}$
    $G \leftarrow 0$
    $W \leftarrow 1$
    for $t \in \{T-1, T-2, ..., 0\}$ do:
        $G \leftarrow R_{t+1} + \gamma G$                          ▷ Recusively compute $G$
        $C(S_t,A_t) \leftarrow C(S_t,A_t) + W$
        $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \frac{W}{C(S_t,A_t)}[G - Q(S_t,A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        if $W = 0$ then:
            break
```

Algorithm 8 Off-policy MC Control

```
Initialisation:
    $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$.
    $C(s,a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$, with ties broken consistently.

loop:
    $b \leftarrow$ any soft policy
    Generate an episode from under $b$: $\{S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T\}$
    $G \leftarrow 0$
    $W \leftarrow 1$
    for $t \in \{T-1, T-2, ..., 0\}$ do:
        $G \leftarrow R_{t+1} + \gamma G$                    ▷ Recusively compute $G$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ with ties broken consistently.
        if $A_t \neq \pi(S_t)$ then:
            break
        else
            $W \leftarrow \frac{W}{b(A_t|S_t)}$
```

Off-Policy MC Control

Algorithm 8 shows an approach to off-policy MC control using weighted importance sampling.

A problem with this algorithm is that it learns only from the tails of episodes where the remaining actions are greedy with respect to the current policy. If non-greedy actions are common, learning will be slow, especially for early actions.

6 Temporal-Difference Learning

Temporal-Difference Learning

Temporal-Difference Learning (TD Learning) is a combination of Monte Carlo and Dynamic Programming ideas. Like MC methods, TD learning uses raw experience without an environmental model but it also bootstraps, using estimates of successor states to improve state estimates.

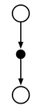
TD Prediction

Monte Carlo methods wait until the return following the visit is known and then use return as a target for $V(S_t)$. For example, *constant-α MC* uses:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)], \quad (21)$$

which is simple and appropriate for non-stationary problems. **TD methods wait only until the next time step, bootstrapping to estimate the reward**,

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]. \quad (22)$$



$TD(0)$ Backup Diagram

This is known as the $TD(0)$ update, an estimate of the value of state as it samples the reward and uses an estimate of the successor state.

Algorithm 9 Tabular TD(0) Prediction

```
Input:
    policy $\pi$ to be evaluated.
    step size $\alpha \in [0,1]$

Initialisation:
    initialise $V(s)$ arbitrarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$. Initalise

for each episode do:
    Generate initial state $S$.
    while episode not terminated do:
        $A \leftarrow \pi(S)$
        Perform action $A$, observe $R, S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
```

The TD error is defined as:

TD Error

$$\delta_t \triangleq R_{t+1} + \gamma V(S_{t+1}) - \gamma V(S). \tag{23}$$

The RD error is the error in the estimate *made at that time*. Note that δ_t is only available at time $t+1$. It can be shown that the Monte Carlo error can be written as a sum of TD errors:

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \tag{24}$$